ORACLE
Academy

Java Foundations

3-2

Numeric Data

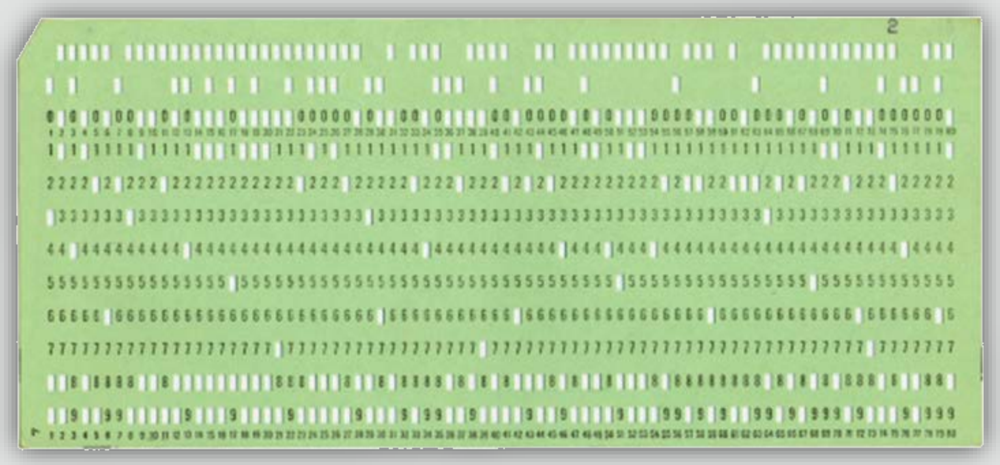
ORACLE
Academy

Objectives

- This lesson covers the following objectives:
  - Differentiate integer data types (byte, short, int, long)
  - Differentiate floating point data types (float, double)
  - Manipulate and do math with numeric data
  - Use parentheses and order of operations

A Bit About Data

- In the early days of computing, data was stored on punch cards



- Each slot had 2 possible states:
  - Punched
  - Not punched

Reading Punch Card Data

- An AutoPiano reads punch cards
- A column represents a key on the piano
- The punch card scrolls through the piano, triggering keys
- Each slot has 2 possible states with 2 possible results:

An 1800s piano roll

State	Result
Punched	Play note
Not punched	Don’t play note

A Bit About Modern Computing

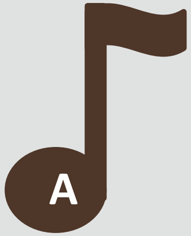
- Modern data processing still needs to represent 2 states:
  - This is interpreted as binary code: **10011101**
  - A single **1** or **0** is called a bit

	AutoPiano	Modern Computing
Bit	Hole punched/Not punched	1/0
Bits are instructions for …	Mechanical components	The processor
Medium	Mechanical	Electro-Magnetism
Bits store data about…	Piano keys	Numbers

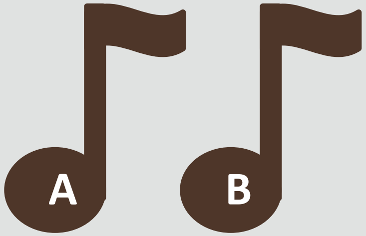
Let's take a closer look at this

Bits of Data

- One AutoPiano key is represented by 1 bit
  - 0: Don’t play
  - 1: Play



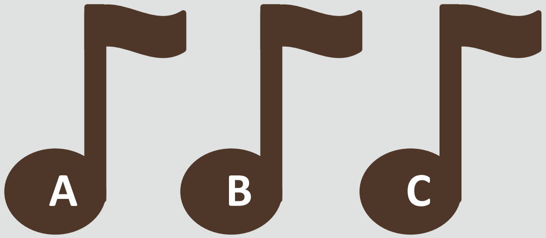
- Two keys require 2 bits
  - There are 4 possible combinations of keys
  - We can calculate this as 2^2



	A key	B key
Silence	0	0
B only	0	1
A only	1	0
Both A and B	1	1

Bigger Bits of Data

- Three keys require 3 bits
  - There are 8 possible combinations of keys
  - We can calculate this as 2^3
- Eight keys require 8 bits
  - There are 256 possible combinations
  - We can calculate this as 2^8



A key	B key	C key
0	0	0
0	0	1
0	1	0
0	1	1
1	0	0
1	0	1
1	1	0
1	1	1

Bits and Bytes

- Eight bits are called a byte
- A Java byte can store 256 possible values
- Possible values are from -128 to 127
  - 128 values below 0
  - 127 values above 0
  - 1 value equal to 0

```
byte x = 127;

byte z = 128;    //Too high
```

Some New Integral Primitive Types

Type	Length	Number of Possible Values	Minimum Value	Maximum Value
Byte	8 bits	2^8, or… 256	-2^7, or… –128	2^7-1, or… 127
short	16 bits	2^{16}, or… 65,535	-2^{15}, or… –32,768	$2^{15}-1$, or… 32,767
int	32 bits	2^{32},or… 4,294,967,296	-2^{31}, or… –2,147,483,648	$2^{31}-1$, or… 2,147,483,647
long	64 bits	2^{64}, or… 18,446,744,073,709,551 ,616	-2^{63}, or… –9,223,372,036, 854,775,808L	$2^{63}-1$, or… 9,223,372,036, 854,775,807L

Note the L

When Will I Use Each Data Type?

`byte`	`short`	`int`	`long`
Never	**Never**	**Often**	**Almost Never**

- byte and short types are used to save memory consumption on older or smaller devices
- But modern desktops contain abundant memory
- Of these 4 types, we’ll mostly use ints in this course

Find x

```
int x = 20;
x = 25;
x = 5 + 3;

System.out.println(x);
```

- x always equals 20 …
  - Until you assign x a different value
- x could be assigned a calculated value

Values for x:	~~20~~	~~25~~	8

Find x

```java
int x = 20;
x = 25;
x = 5 + 3;
x = x + 1;
x += 1;
x++;
System.out.println(x);
```

- x could be assigned a new value based on its current value:
  - Java provides the shorthand += operator to do this
  - Adding 1 to a variable is so common that Java provides the shorthand ++ operator

Values for x: ~~20~~ ~~25~~ ~~8~~ ~~9~~ ~~49~~ 11

Find x Again

- x could be assigned the value of another variable:
  - Changing y doesn't change x
  - y and x are separate variables

```
int y = 20;
int x = y;
y++;

System.out.println(x);
System.out.println(y);
```

- Output:

x	20
y	21

Standard Mathematical Operators

Purpose	Operator	Example	Comments

Addition	+	`int sum = 0;`
		`int num1 = 10;`
		`int num2 = 2;`
		`sum = num1 + num2;`
			If num1 is 10 and num2 is 2, sum is 12

Subtraction	–	`int diff = 0;`
		`int num1 = 10;`
		`int num2 = 2`
		`diff = num1 – num2;`
			If num1 is 10 and num2 is 2, diff is 8

Standard Mathematical Operators

Purpose	Operator	Example	Comments

Multiplication	*	`int prod = 0;`
		`int num1 = 10;`
		`int num2 = 2;`
		`prod = num1 * num2;`
			If num1 is 10 and num2 is 2, prod is 20

Division	/	`int quot = 0;`
		`int num1 = 31;`
		`int num2 = 2;`
		`quot = num1 / num2;`
			If num1 is 31 and num2 is 6, quot is 5
			
			The remainder portion is discarded
			
			Note:
			Division by 0 returns an error

Why?

Since `int` data types are whole numbers only, the decimal remainder will be discarded. You will see how to change this behavior later in this lesson.

Using Java Shorthand Operators to Make Assignments

- A shorthand operator is a shorter way to express something that is already available in the Java programming language

Purpose	Operator	Shorthand Operator Examples	Equivalent Construct	Result

Add to and assign	`+=`	`int a = 6;`
		`int b = 2;`
		`a += b;`
			`int a = 6;`
			`int b = 2;`
			`a = a + b;`
				a = 8

Subtract from and assign	`-=`	`int a = 6;`
		`int b = 2;`
		`a -= b;`
			`int a = 6;`
			`int b = 2;`
			`a = a - b;`
				a = 4

Using Java Shorthand Operators to Make Assignments

Purpose	Operator	Shorthand Operator Examples	Equivalent Construct	Result

Multiply by and assign	`*=`	`int a = 6;`
		`int b = 2;`
		`a *= b;`
			`int a = 6;`
			`int b = 2;`
			`a  = a * b;`
				a = 12

Divide by and assign	`/=`	`int a = 6;`
		`int b = 2;`
		`a  /= b;`
			`int a = 6;`
			`int b = 2;`
			`a  = a / b;`
				a = 3

Get remainder and assign	`%=`	`int a = 6;`
		`int b = 2;`
		`a %= b;`
			`int a = 6;`
			`int b = 2;`
			`a  = a % b;`
				a = 0

Modulus Operator

Purpose	Operator	Example	Comments

Remainder	%
	
	(modulus)
		`num1 = 31;`
		`num2 = 6;`
		
		`mod = num1 % num2;`
		
		(mod = 1)
			Modulus finds the remainder of the first number divided by the second number.
			
			5 R 1
			6 | 31
			30
			----- 	
			1
			Modulus always gives an answer with the same sign as the first operand.

Increment and Decrement Operators (++ and - -)

- The long way:
  - `age = age + 1;`

  or
  - `count = count – 1;`

- The short way:
  - `age++;`

  or
  - `count--;`

More on Increment and Decrement Operators

Operator	Purpose	Example

++	Pre-increment (++variable)	`int id = 6;`
		`int newId = ++id;`
		id is 7, newId is 7

	Post-increment (variable++)	`int id = 6;`
		`int newId = id++;`
		id is 7, newId is 6

- -	Pre-decrement (--variable)	(Same principle applies)
	Post-decrement (variable--)	

Increment and Decrement Operators (++ and ––)

```
int count=15;
int a, b, c, d;
a = count++;
b = count;
c = ++count;
d = count;
System.out.println(a + ", " + b + ", " + c + ", " + d);
```

- Output:

```
15, 16, 17, 17
```

Exercise 1, Part 1

- Create a new project and add the `Chickens01.java` file to the project
- Read this story and calculate/print the totalEggs collected between Monday and Wednesday:
  - Farmer Brown's chickens always lay eggsPerChicken eggs precisely at noon, which he collects that day
  - On Monday, Farmer Brown has chickenCount chickens
  - On Tuesday morning, Farmer Brown gains 1 chicken
  - On Wednesday morning, a wild beast eats half the chickens!
  - How many eggs did Farmer Brown collect if he starts with …
    - eggsPerChicken = 5, chickenCount = 3
    - eggsPerChicken = 4, chickenCount = 8

Exercise 1, Part 2

- Your program should produce the following output:

```
45    First scenario
84    Second scenario
```

Integer Division Deception

- The wild beast ate half the chickens
- When we divide 9 chickens in half, Java thinks 9/2 = 4
  - But 9/2 = 4.5
  - Shouldn't Java round up to 5?
  - What's going on here?

Java Division

- Java integers aren't rounded
- Java integers are truncated, meaning any numbers after the decimal point are removed

```
int x = 9/2;
System.out.println(x);  //prints 4
```

- We need other data types if we have scenarios that require floating point precision!

Floating Point Primitive Types

Type	Float Length	When will I use this?
`float`	32 bits	Never
`double`	64 bits	Often

Double the precision of a float

- Example:
  - `public float pi  = 3.141592F;`
  - `public double pi = 3.141592;`

Note the F

Double Deception

- The original problem:

```
int x = 9/2;
System.out.println(x);  //prints 4
```

- Shouldn't a double x fix this?

```
double x = 9/2;
System.out.println(x);  //prints 4.0
```

  - No?!?!
  - Why not?

Double Deception

```
double x = 9/2;
System.out.println(x);  //prints 4.0
```

- Java solves the expression, truncates the .5, and then turns the answer into a double
- The expression contains only ints, Java won't allocate the additional memory that doubles require until it absolutely has to
  - Solution: Include a double in the expression

```
double x = 9/2.0;
System.out.println(x);  //prints 4.5
```

One Final Note

- Declare a variable with the final keyword to make its value unchangeable (immutable)

```java
final double PI = 3.141592;
PI = 3.0;          //Not Allowed
```

- Java complains if you try to change a final variable's value
- Final variable naming conventions:
  - Capitalize every letter
  - Separate words with an underscore
    - MINIMUM_AGE
    - SPEED_OF_LIGHT

Exercise 2, Part 1

- Create a new project and add the `Chickens02.java` file to the project
- Read this story and calculate/print the required values:
  - On Monday, Farmer Fred collects 100 eggs
  - On Tuesday, Farmer Fred collects 121 eggs
  - On Wednesday, Farmer Fred collects 117 eggs
  - What is the dailyAverage of eggs collected?
  - How many eggs could be expected in a 30-day monthlyAverage?
  - If an egg can be sold for a profit of $0.18, what is Farmer Fred's total monthlyProfit for all eggs?

Exercise 2, Part 2

- Your program should produce the following output:

```
Daily Average:   112.66666666666667
Monthly Average: 3380.0
Profit:          $608.4
```

Parentheses in Mathematical Expressions

- This expression without parentheses …

```
int x = 10 +20 +30 / 3;                    //x=40
```

- Is just like writing this expression with parentheses:

```
int x = 10 +20 +(30 / 3);                  //x=40
```

- If you want to find an average, use parentheses like this:

```
int x = (10 +20 +30) / 3;                  //x=20
```

Operator Precedence

- Here’s an example of the need for rules of precedence:

```
int x = 25 - 5 * 4 / 2 - 10 + 4;
```

- Is the answer 34 or 9?

- Add parenthesis to enforce precedence

Rules of Precedence

- Operators within a pair of parentheses
- Increment and decrement operators (++ or - -)
- Multiplication and division operators, evaluated from left to right
- Addition and subtraction operators, evaluated from left to right
- If operators of the same precedence appear successively, the operators are evaluated from left to right

Using Parentheses

- Expression are evaluated with the rules of precedence
- However, you should use parentheses to provide the intended structure
- Examples:

```
int x = (((25 - 5) * 4) / (2 - 10)) + 4;
int x = ((20 * 4) / (2 - 10)) + 4;
int x = (80 / (2 - 10)) + 4;
int x = (80 / -8) + 4;
int x = -10 + 4;
int x = -6;
```

Summary

- In this lesson, you should have learned how to:
  - Differentiate integer data types (byte, short, int, long)
  - Differentiate floating point data types (float, double)
  - Manipulate and do math with numeric data
  - Use parentheses and order of operations

ORACLE
Academy

ORACLE
Academy